Implementation Document

for

FinXpert

Prepared by

Group 17: **Group Name:** Tech Titans

Dwij Om Oshoin	220386	dwij22@iitk.ac.in
Karthik S Pillai	220502	karthiksp22@iitk.ac.in
Rahul Mahato	220859	mrahul22@iitk.ac.in
Archit Atrey	220195	aarchit22@iitk.ac.in
Sumit Kumar	221102	sumitkumar22@iitk.ac.in
Ayush Kumar Yadav	220267	ayushku22@iitk.ac.in
Piyush Meena	220768	piyushm22@iitk.ac.in
Madhav Khetan	220596	madhavk22@iitk.ac.in
Aniket Kumar Choudhary	220140	aniketkc22@iitk.ac.in
Devank Saran Prajapati	220339	devanks22@iitk.ac.in

Course: CS253

Mentor TA: *Naman Baranwal*

Date: 28/03/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1 Implementation Details

Frontend

The FinXpert frontend is built using React.js with Vite.js, creating a modern and efficient financial management platform. We chose this stack for its:

1. Modern Development Experience

- Lightning-fast development with Vite.js
- Hot Module Replacement for instant updates
- Optimized production builds

2. Component-Based Architecture

- Reusable UI components using Material-UI
- Consistent design system
- Modular code structure

3. Rich User Interface

- Interactive data visualizations with Nivo charts
- Smooth animations with Framer Motion
- Responsive design for all devices

4.Project Structure

```
frontend/
├── src/
│   ├── assets/        # Static assets
│   ├── components/    # Reusable components
│   ├── scenes/        # Page components
│   ├── context/       # React context
│   ├── state/         # Redux store
│   ├── utils/         # Utility functions
│   ├── animations/    # Animation configs
│   ├── theme.js       # MUI theme
│   ├── App.jsx        # Main component
│   └── main.jsx       # Entry point
```

5. Key Features

- Interactive financial dashboards
- Real-time chat interface
- Data visualization charts
- Form validation
- Responsive navigation
- Dark/Light theme
- Loading animations

6. Technical Stack

- React.js with Vite.js
- Material-UI for components
- Redux for state management
- Socket.IO for real-time features
- Formik for form handling
- Nivo for data visualization
- Framer Motion for animations

7. Development Tools

- ESLint for code quality
- TypeScript support
- Testing with Jest
- Development server
- Production optimization

8.Performance & Security

- Code splitting
- Optimized assets
- Secure form handling
- Protected routes
- API security

This modern frontend stack ensures:

- Fast and responsive UI
- Consistent user experience
- Easy maintenance
- Efficient development
- High performance
- Professional appearance
- Accessibility compliance

Backend

FinXpert leverages Node.js and a microservices architecture to deliver high performance, scalability, and real-time financial insights. Key technologies and advantages include:

● Microservices

The microservices used for deployment are:

1. **User Authentication Service**
2. **Dashboard Service**
3. **Expense Service**
4. **Income Service**
5. **Financial Data Analysis Service**
6. **Smart AI Service**

Docker is used to containerize each microservice, ensuring consistency across different environments. Each service runs inside its own container with all dependencies included. Kubernetes is used to orchestrate the deployment, scaling, and management of microservices across multiple machines. It automates tasks like **s**ervice discovery, load balancing, self-healing, and rolling updates.

● Microservices Benefits

1. Decoupled Services: Independent deployment of User, Expense, and AI services.
2. Resilience: Failure in one service (e.g., Income) doesn't crash the entire system.
3. Scalability: Kubernetes auto-scales high-traffic services like Financial Data.

● Microservices logic

1.) **User Service**
   (i) Handles user management and authentication
   (ii) Manages user profiles, registration, and login
   (iii) Responsible for user-related operations and security
2.) **Smart AI Service**
   (i) Provides AI-powered financial insights and recommendations
   (ii) Handles machine learning models for financial predictions
   (iii) Processes and analyzes financial data to generate intelligent insights
3.) **Orchestrator Service**
   (i) Acts as the central coordinator for all microservices
   (ii) Manages service-to-service communication
   (iii) Handles workflow orchestration and service coordination
4.) **Income Service**

(i) Manages all income-related operations
(ii) Handles income tracking, categorization, and analysis
(iii) Processes income data and generates income-related reports

5.) **Financial Data Service**
(i) Manages core financial data operations
(ii) Handles data storage and retrieval for financial information
(iii) Provides financial data aggregation and processing.

6.) **Expense Service**
(i) Manages expense tracking and categorization
(ii) Handles expense-related operations and analysis
(iii) Processes expense data and generates expense reports

7.) **Event Bus Service**
(i) Manages asynchronous communication between services
(ii) Handles event publishing and subscription
(iii) Ensures reliable message delivery between microservices

8.) **Dashboard Service**
(i) Manages the dashboard UI and data presentation
(ii) Handles dashboard customization and user preferences
(iii) Aggregates data from other services for dashboard display.

- ## Node.js with Express.js

1. Lightweight and asynchronous, ideal for handling concurrent API requests.
2. Event-driven architecture aligns perfectly with Kafka for real-time data processing.
3. The NPM ecosystem provides access to 1.3 M+ packages (e.g., `kafka js`, `jsonwebtoken`).

- ## Key Backend Libraries:

Library	**Purpose**
KafkaJs	Kafka event streaming
Mongoose	MongoDB ODM (for Expense Service)
jsonwebtoken	Secure JWT authentication
Rate-limiter	OTP request throttling

- ## Security:

1. **OTP Rate Limiting**: Prevents brute-force attacks (3 attempts/hour).
2. **JWT Encryption**: Tokens signed with HS256 algorithm.

3. **Kubernetes Secrets**: Stores DB credentials and API keys securely.

Database

FinXpert uses a hybrid database system: MongoDB for unstructured financial data and PostgreSQL for structured user data.

- **MongoDB (Expense/Income Services)**
  - **Schema Design:** Ideal for storing variable financial data (e.g., cognitive triggers).
  - **Indexing:** Uses userId and timestamp indexes to accelerate queries.
  - **Transactions:** Ensures atomicity for multi-document updates.
- **PostgreSQL (User Service)**
  - **ACID Compliance:** Critical for maintaining data integrity (e.g., authentication records).

Benefits of This Approach Data Type Optimization PostgreSQL:

Perfect for structured, relational data MongoDB: Ideal for flexible, document-based data Performance Benefits PostgreSQL: Fast complex queries and joins MongoDB: High write performance and scalability Development Benefits PostgreSQL: Strong data consistency MongoDB: Rapid development and schema evolution Business Benefits Better data organization Improved system performance Enhanced user experience Cost-effective solution This hybrid approach allows FinXpert to handle both structured and unstructured data efficiently, providing the best of both worlds for different types of financial data management.

Key Advantages of FinXpert's Stack :

Aspect	Benefit
Microservices	Independent scaling (e.g., AI service can handle 10K+ NLP requests/hour)
Kafka	Guaranteed message delivery; decouples services
React	Reusable components (e.g., '<FinancialChart >' used in 5+ views)
MongoDB	Flexible schema accommodates new fields (e.g., 'mood', 'cognitiveTriggers')

2 Codebase

Github Repositories:

- https://github.com/RahulMahato23/FinXpert
- http://github.com/RahulMahato23/FinXpertRun

Code Structure

FinXpert is built using a microservices architecture with Node.js, which segregates the project into different services for various major functionalities. This ensures a well-structured and easily accessible codebase. The project has been divided into the following microservices:

User Service: Handles user management, authentication, and profile management. Users can register, login, and manage their account settings.

Smart AI Service: Provides AI-powered financial insights and recommendations using OpenAI integration.

Orchestrator Service: Manages service-to-service communication and workflow orchestration.

I**ncome Service**: Handles income tracking, categorization, and analysis.

Financial Data Service: Manages core financial data operations and aggregation.

Expense Service: Handles expense tracking, categorization, and analysis.

Event Bus Service: Manages asynchronous communication between services using Kafka.

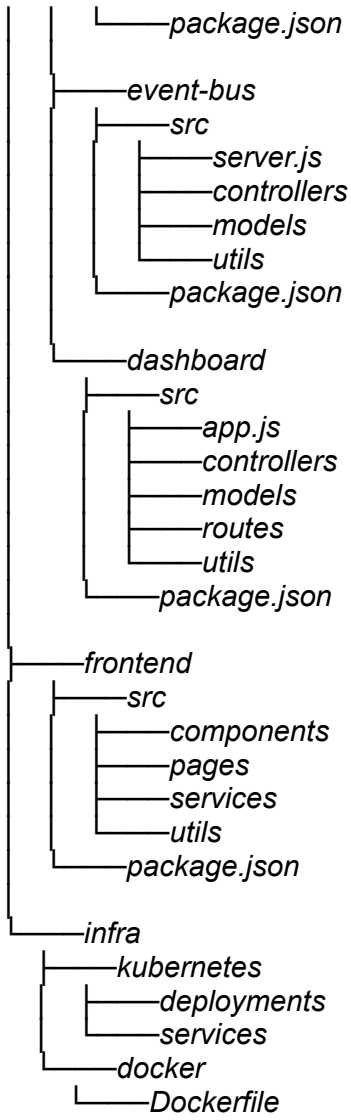
Dashboard Service: Manages the dashboard UI and data presentation.

Directories

```
└──FinXpert-main
   package.json
   package-lock.json
   skaffold.yaml
   docker-script.sh
   updateScript.sh
   monitor-pods.sh
   README.md
   LICENSE

├──services
   ├──user
      ├──src
         ├──app.js
```

```
        │   │   ├── config
        │   │   │   └── migrate.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
        │   └── package.json
        │
        ├── smart-ai
        │   ├── src
        │   │   ├── app.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
        │   └── package.json
        │
        ├── orchestrator-service
        │   ├── src
        │   │   ├── app.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
        │   └── package.json
        │
        ├── income
        │   ├── src
        │   │   ├── app.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
        │   └── package.json
        │
        ├── financial-data
        │   ├── src
        │   │   ├── app.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
        │   └── package.json
        │
        ├── expense
        │   ├── src
        │   │   ├── app.js
        │   │   ├── controllers
        │   │   ├── models
        │   │   ├── routes
        │   │   └── utils
```



How to Run the Application

Prerequisites

- A PostgreSQL and MongoDB Database (multiple if needed)
- Docker and Kubernetes must be installed
- Skaffold must be installed in the system
- You must have an OpenAI API Key for AI features

Extract the `development.zip` and open the folder in your IDE, then follow these steps.

Steps

1. Secrets Deployment

1. Navigate to the `secrets/secrets` directory inside `development.zip`.

2. Modify the environment variables in each file as per the given instructions.
3. Run the following script in the parent `secrets` directory: ./secret-deploy-script.sh
4. If redeployment is required, delete existing secrets and redeploy using: kubectl delete secrets --all -n finxpert-dev

Then, deploy the secrets again.

2. Deploying Kafka

1. Create a Persistent Volume Claim (PVC) for Kafka:
   kubectl apply -f ./k8s/kafka/pvc.yaml
   Wait 10 seconds.
2. Deploy Zookeeper:
   kubectl apply -f ./k8s/kafka/zookeeper-depl.yaml
   Wait 20 seconds and ensure the Zookeeper pod is running.
3. Deploy Kafka:
   kubectl apply -f ./k8s/kafka/kafka-depl.yaml
   Wait 60 seconds to ensure Kafka is running.
4. Run the Kafka topic creator job: kubectl apply -f ./k8s/kafka/kafka-create-topics-job.yaml

3. Deploy the Ingress Controller

1. Install `ingress-nginx` in your Kubernetes cluster:
   kubectl apply -f "https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml"
2. *(OPTIONAL: Only if `skaffold dev` shows ingress-related issues)*

4. Change the Hosts File

Required for `finxpert.com` to point to `127.0.0.1`.

For Windows:

1. Go to `C:\Windows\System32\drivers\etc`.
2. Open the `hosts` file in a text editor with Administrator permissions.
3. Add the following line at the end:
   127.0.0.1 finxpert.com
4. Save the file. *(A system reboot may be required.)*

For Linux/macOS:

- Search online for steps to modify the hosts file for your OS.

5. Starting the Backend

1. Navigate to `/services/user/src/migrations/up`.
2. Run the `.sql` files on your user database using your preferred method.

3. Go to the project root directory and run:
   skaffold dev
   Wait for the process to complete.

6. Running the Frontend

1. Create a `.env` file in the `/frontend` directory.
2. .Add the following variables:
   VITE_REACT_APP_API_BASE_URL=http://finxpert.com/api,
   VITE_REACT_APP_BASE_URL=http://finxpert.com
3. Start the frontend using: npm run dev

The application should now be running successfully.

3 Completeness

Provide the details of the part of the SRS that have been completed in the implementation.

Provide the future development plan by listing down the features that will be added in the (may be hypothetical) future versions.

Functionalities of FinXpert that have been Completed:

1. Core Functionalities

(i) **Expense & Income Tracking** – Users can log, update, and categorize financial transactions. Transactions are securely stored in MongoDB with event-driven updates using Kafka.

(ii) **Interactive Dashboard** – A real-time financial dashboard provides visual insights into spending trends, income distribution, and financial summaries.

(iii)**AI/NLP-Powered Chat Assistant** – Users can interact via natural language to add expenses, request financial summaries, and receive personalized recommendations.

(iv) **Financial Data Analysis** – Aggregates user transactions, identifies spending patterns, and generates AI-driven insights.

2. Security & Authentication

(i) OTP-Based Secure Login – Users authenticate via one-time password (OTP), with rate-limiting to prevent abuse.

(ii) **Role-Based Access Control (RBAC)** – Ensures restricted access based on user roles (Admin/User).

(iii) **Data Encryption** – All sensitive financial data is encrypted both at rest and in transit.

3. System Architecture & Performance

(i) **Microservices Architecture** – The platform is built on **Node.js, Express.js, Kafka, and Kubernetes**, ensuring modularity and scalability.

(ii) **Event-Driven Processing** – Services communicate asynchronously via Kafka to handle high-volume transactions efficiently.

(iii) **Scalability & Deployment** – Deployed on **Google Cloud Platform (GCP)** using Kubernetes for high availability and auto-scaling.

4. User Experience Enhancements

(i) **Responsive Web Application** – Developed using **React.js and Redux**, optimized for web and mobile devices.

(ii) **Real-Time Updates** – Financial data, AI responses, and dashboard insights refresh dynamically within seconds.

(iii) **Budget Alerts & Notifications** – Users receive alerts when approaching their spending limits.

Status and Future Development Plan of each Features:

Features	Status	Future Development Plan
User Account Management	Completed	Use mobile number to login
Expense Management	Completed	Notify user after every transaction
Income Management	Completed	Implement **automated categorization** of income sources.
Dashboard Functionality	Completed	**Unified Financial Ledger** → Store all transactions in a single structured format.
Smart AI Assistant	Completed	**Voice-Enabled AI Assistant** → Query financial data via voice commands
Security and Privacy	Completed	**Biometric Authentication** → Fingerprint or Face ID login for mobile app
Frontend Functionality	Completed	Toggle dark and light mode
Financial Data Analysis	Completed	Connect with banks, e-wallets, and crypto transactions

Appendix A - Group Log

All the group members were in constant touch with each other and the TA through the WhatsApp group. Suggestions were given by the TA regarding some corrections and elaboration in the architecture diagram, class diagrams, and state diagram. All those suggestions have been seriously incorporated into the document. Work was divided, and team meetings were held regularly.

Meeting Log Table

Date	Members Present	Description
26/2/2025	All Group Members	Distribution of work and decided the utilities and libraries to use. Discussion on frontend progress.
1/3/2025	TA and All Group Members	Addressed and resolved TA's questions and suggestions regarding our software.
2/3/2025	All Group Members	Discussion on initial models and views for Login, Signup Pages.
3/3/2025	All Group Members	Discussion on initial models and views for Guest Room Booking.
4/3/2025	All Group Members	Discussion on initial models and views for Complaint Management.
5/3/2025	All Group Members	Databasing of software.
6/3/2025	All Group Members	Discussion on final models and views for all pages.
10/3/2025	All Group Members	Finalized overall software backend as well as frontend, also modified to achieve better results.
12/3/2025	TA and All Group Members	Checked on software functionality, performance, reliability, accessibility, efficiency, correctness, usability, and integrity.
15/3/2025	All Group Members	Locating and fixing bugs in the software.
20/3/2025	TA Group Members	Locating and fixing bugs in the software.
25/3/2025	All Group Members	Worked on the creation of the Implementation Document.